Homework #1 - SQL

OVERVIEW

The first homework is to construct a set of SQL queries for analysing a dataset that will be provided to you. For this, you will look into IMDB data. This homework is an opportunity to: (1) learn basic and certain advanced SQL features, and (2) get familiar with using a full-featured DBMS, SQLite, that can be useful for you in the future.

This is a single-person project that will be completed individually (i.e., no groups).

- **Release Date:** Jan 23, 2023
- **Due Date:** Feb 03, 2023 @ 11:59pm

SPECIFICATION

The homework contains 10 questions in total and is graded out of 100 points. For each question, you will need to construct a SQL query that fetches the desired data from the SQLite DBMS. It will likely take you approximately 6-8 hours to complete the questions.

Placeholder Folder

Create the placeholder submission folder with the empty SQL files that you will use for each question:

```
$ mkdir placeholder
$ cd placeholder
$ touch \
  q1_sample.sql \
  q2_not_the_same_title.sql \
  q3_longest_running_tv.sql \
  q4_directors_in_each_decade.sql \
  q5_german_type_ratings.sql \
  q6_who_played_a_batman.sql \
  q7_born_with_prestige.sql \
  q8_directing_rose.sql \
  q9_ode_to_death.sql \
  q10_all_played_by_leo.sql
$ cd ..
```

After filling in the queries, you can compress the folder by running the following command:

```
$ zip -j submission.zip placeholder/*.sql
```

The `-j` flag lets you compress all the SQL queries in the zip file without path information. The grading scripts will **not** work correctly unless you do this.

INSTRUCTIONS

Setting Up SQLite

You will first need to install SQLite on your development machine.

will **not** support the SQL features that you need to complete this assignment.

Install SQLite3 on Ubuntu Linux

Please follow the instructions.

Install SQLite3 on Mac OS X

On Mac OS Leopard or later, you don't have to! It comes pre-installed. You can upgrade it, if you absolutely need to, with Homebrew.

Load the Database Dump

1. Check if `sqlite3` is properly working by following this tutorial.
2. Download the database dump file:

   ```
   $ wget https://15445.courses.cs.cmu.edu/fall2022/files/imdb-cmudb2022.db.gz
   ```

   Check its MD5 checksum to ensure that you have correctly downloaded the file:

   ```
   $ md5sum imdb-cmudb2022.db.gz
   d7cdf34f4ba029597c3774fc96bc3519  imdb-cmudb2022.db.gz
   ```

3. Unzip the database from the provided database dump by running the following commands on your shell. Note that the database file be **836MB** after you decompress it.

   ```
   $ gunzip imdb-cmudb2022.db.gz
   $ sqlite3 imdb-cmudb2022.db
   ```

4. We have prepared a random sample of the original dataset for this assignment. Although this is not required to complete the assignment, the complete dataset is available by following the steps here.
5. Check the contents of the database by running the `.tables` command on the `sqlite3` terminal. You should see **6 tables**, and the output should look like this:

   ```
   $ sqlite3 imdb-cmudb2022.db
   SQLite version 3.31.1
   Enter ".help" for usage hints.
   sqlite> .tables
   akas      crew      episodes  people    ratings   titles
   ```

6. Create indices using the following commands in SQLite:

   ```
   CREATE INDEX ix_people_name ON people (name);
   CREATE INDEX ix_titles_type ON titles (type);
   CREATE INDEX ix_titles_primary_title ON titles (primary_title);
   CREATE INDEX ix_titles_original_title ON titles (original_title);
   CREATE INDEX ix_akas_title_id ON akas (title_id);
   CREATE INDEX ix_akas_title ON akas (title);
   CREATE INDEX ix_crew_title_id ON crew (title_id);
   CREATE INDEX ix_crew_person_id ON crew (person_id);
   ```

Check the schema

Get familiar with the schema (structure) of the tables (what attributes do they contain, what are the primary and foreign keys). Run the `.schema $TABLE_NAME` command on the `sqlite3` terminal for each table. The output should look like the example below for each table.

PEOPLE

```
sqlite> .schema people
CREATE TABLE people (
```

```
);
CREATE INDEX ix_people_name ON people (name);
```

Contains details for a person. For example, this is a row from the table:

```
nm0000006|Ingrid Bergman|1915|1982
```

To breakdown the row, the field `person_id` corresponds to "nm0000006", `name` corresponds to "Ingrid Bergman", `born` corresponds to "1981", and `died` corresponds to "1982".

TITLES

```
sqlite> .schema titles
CREATE TABLE titles (
  title_id VARCHAR PRIMARY KEY,
  type VARCHAR,
  primary_title VARCHAR,
  original_title VARCHAR,
  is_adult INTEGER,
  premiered INTEGER,
  ended INTEGER,
  runtime_minutes INTEGER,
  genres VARCHAR
);
CREATE INDEX ix_titles_type ON titles (type);
CREATE INDEX ix_titles_primary_title ON titles (primary_title);
CREATE INDEX ix_titles_original_title ON titles (original_title);
```

Contains details of a title. For example, this is a row from the table:

```
tt0088763|movie|Back to the Future|Back to the Future|0|1985||116|Adventure,Comedy,Sci-Fi
```

For this assignment, we will focus on the fields `title_id` (e.g. "tt0088763"), `type` (e.g. "movie"), `primary_title` (e.g. "Back to the Future"), `premiered` (e.g. "1985"), `ended` (in this case `NULL`) and `genres` (e.g. "Adventure,Comedy,Sci-Fi").

Titles may also be referred to as "works" in the assignment specification.

AKAS

```
sqlite> .schema akas
CREATE TABLE akas (
  title_id VARCHAR, -- REFERENCES titles (title_id),
  title VARCHAR,
  region VARCHAR,
  language VARCHAR,
  types VARCHAR,
  attributes VARCHAR,
  is_original_title INTEGER
);
CREATE INDEX ix_akas_title_id ON akas (title_id);
CREATE INDEX ix_akas_title ON akas (title);
```

This table contains the alternate titles for the dubbed movies. Note that `title_id` in this table corresponds to `title_id` in titles. For example, this is a row in the table:

```
tt0015648|El acorazado Potemkin|XSA|es|imdbDisplay||0
```

For this assignment, we will not use the fields `region`, `attributes` or `is_original_title`.

Note that title_id in this table corresponds to title_id in titles.

CREW

```
sqlite> .schema crew
CREATE TABLE crew (
  title_id VARCHAR, -- REFERENCES titles (title_id),
  person_id VARCHAR, -- REFERENCES people (person_id),
  category VARCHAR,
  job VARCHAR,
  characters VARCHAR
);
CREATE INDEX ix_crew_title_id ON crew (title_id);
CREATE INDEX ix_crew_person_id ON crew (person_id);
```

```
tt0000886|nm0609814|actor|||["Hamlet"]
```

For this assignment, we will not use the fields `job`. When considering the role of an individual on the crew, refer to the field `category`. Note that a cast member can play mutliple `characters` under a title.

Note that title_id corresponds to title_id in titles and person_id corresponds to person_id in people.

RATINGS

```
sqlite> .schema ratings
CREATE TABLE ratings (
  title_id VARCHAR PRIMARY KEY, -- REFERENCES titles (title_id),
  rating FLOAT,
  votes INTEGER
);
```

Contains the ratings for each title. For example, this is a row from the table:

```
tt0000803|6.0|8
```

Note that title_id in this table corresponds to title_id in titles.

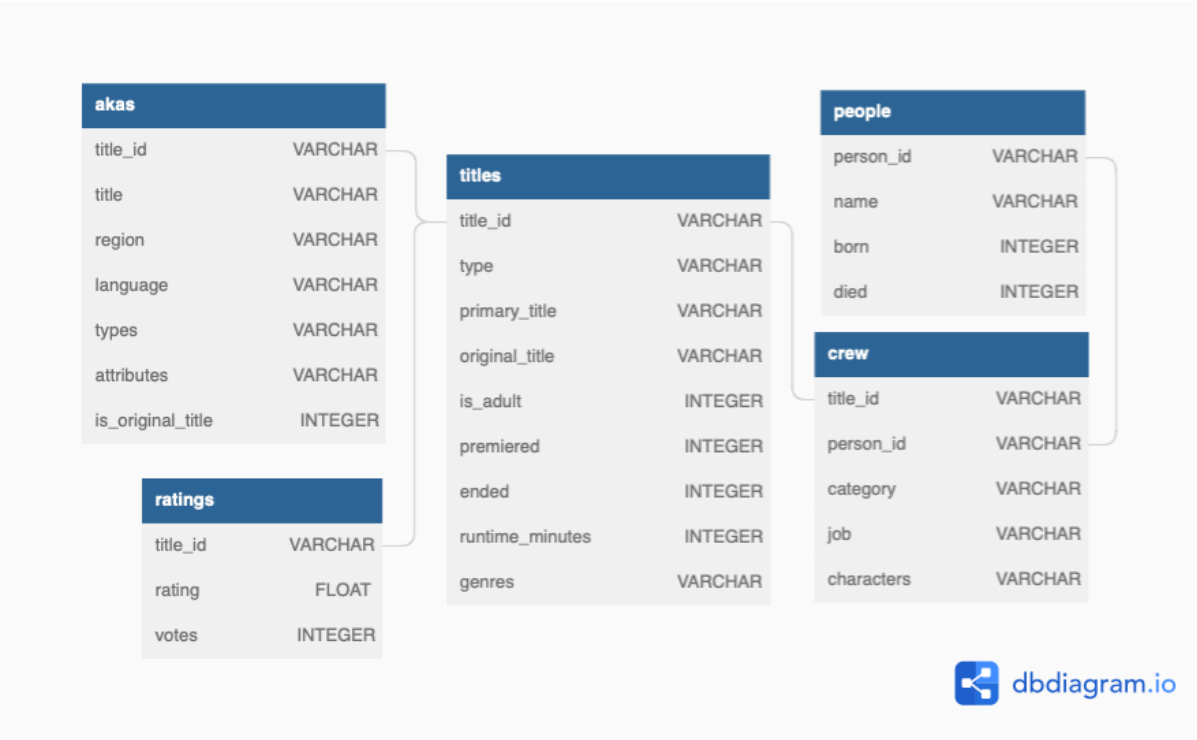
EPISODES

While the table `episodes` is included in our sample dataset, you should not need to reference this table.

Sanity Check

Count the number of rows in the `titles` table

```
sqlite> select count(*) from titles;
1375462
```



Construct the SQL Queries

Now, it's time to start constructing the SQL queries and put them into the placeholder files.

the formatting of our auto-grading script.
Details: List the first ten roles of the cast crew members ordered alphabetically.

Answer: Here's the correct SQL query and expected output:

```
sqlite> SELECT DISTINCT(category)
   ...> FROM crew
   ...> ORDER BY category
   ...> LIMIT 10;
actor
actress
archive_footage
archive_sound
cinematographer
composer
director
editor
producer
production_designer
```

You should put this SQL query into the appropriate file (`q1_sample.sql`) in the submission directory (`placeholder`).

Q2 [5 points] (q2_not_the_same_title):

Find the 10 Action movies with the newest premiere date whose original title is not the same as its primary title.
Details: Print the premiere year, followed by the two titles in a special format. The column listing the two titles should be in the format of `primary_title (original_title)` Note a work is `Action` even if it is categorized in multiple genres, as long as `Action` is one of the genres. Also note that it's possible for the premiered year to be in the future. If multiple movies premiered in the same year, order them alphabetically. Your first row should look like this:

`2027|The Adventures of Tintin: Red Rackham's Treasure (Untitled Adventures of Tintin Sequel)`

Q3 [5 points] (q3_longest_running_tv):

Find the top 20 longest running tv series.
Details: Print the title and the years the series has been running for. The series must have a non `NULL` premiered year. If the ended date is `NULL` , assume it to be the current year (2023). If multiple tv series have been running the same number of years, order them alphabetically.
Print the top 20 results.
Your output should have the format: `TITLE|YEARS_RUNNING`
Your first row should look like this: `Looney Tunes|93`

Q4 [10 points] (q4_directors_in_each_decade):

List the number directors born in each decade since 1900.
Details: Print the decade in a fancier format by constructing a string that looks like this: `1990s` .
Order the results by decade.
Your output should look like this: `DECADE|NUM_DIRECTORS`
Your first row should look like this: `1900s|376`

Q5 [10 points] (q5_german_type_ratings):

Compute statistics about different type of works that has a German title.

output by the average rating of each title type.
Your output should have the format: `TITLE_TYPE|AVG_RATING|MIN_RATING|MAX_RATING`
Your first row should look like this: `movie|6.65|3.4|8.2`

Q6 [10 points] (q6_who_played_a_batman):

List the 10 highest rated actors who played a character named "Batman".
Details: Calculate the actor rating by taking the average rating of all their works. Return both the name of the actor and their rating and only list the top 10 results in order from highest to lowest rating. Round average rating to the nearest hundredth.
Make sure your output is formatted as follows: `Kayd Currier|8.05`

Q7 [15 points] (q7_born_with_prestige):

List the number of actors or actress who were born on the year that "The Prestige" was premiered.
Details: Print only the total number of actors born that year. For this question, determine distinct people by their `person_id`, not their names. Do not hard code the query.

Q8 [15 points] (q8_directing_rose.sql):

Find all the directors who have worked with an actress with first name "Rose".
Details: Print only the names of the directors in alphabetical order. Each name should only appear once in the output.
Your first row should look like this: `Aimé Forget`

Q9 [15 points] (q9_ode_to_the_dead):

List the longest work for the first 5 cast members sorted by year of death for each type of role.
Details: For each role, find the first 5 artists sorted by year of death and print their work with the longest runtime. The final output should be sorted alphabetically by category, then by year of death of the artist and finally by alphabetical order of artist name. If an artist has multiple works with the the same longest runtime, pick the one with the smallest `title_id`. Your output should look like this: `CATEGORY|NAME|DEATH YEAR|LONGEST WORK TITLE|WORK RUNTIME|CATEGORY RANK` Your first row should look like this: `actor|Verner Clarges|1911|A Summer Idyll|17|1`

Note: Omit artists that do not have work with a non-null runtime.

Q10 [15 points] (q10_all_played_by_leo):

List all the unique characters Leonardo DiCaprio (born in 1974) ever played as a string of comma-separated values in alphabetical order of the character names.
Details: Find all the unique characters played by Leonardo DiCaprio and order them alphabetically. Exclude those characters containing the character sequence `Self`. Print a single string containing all these characters separated by commas.
Hint: You might find Recursive CTEs useful.
Note: Watch out for the format of the `characters` field. An actor can play more than one characters in a single work.

Each submission will be graded based on whether the SQL queries fetch the expected sets of tuples from the database. Note that your SQL queries will be auto-graded by comparing their outputs (i.e. tuple sets) to the correct outputs. For your queries, the **order** of the output columns is important; their names are not.

LATE POLICY

See the late policy in the syllabus.

SUBMISSION

We use the Autograder from Gradescope for grading in order to provide you with immediate feedback. After completing the homework, you can submit your compressed folder `submission.zip` (only one file) to Gradescope:

- **https://www.gradescope.com/courses/485657**

Important: Use the Gradescope course code announced on Piazza.

We will be comparing the output files using a function similar to `diff`. You can submit your answers as many times as you like.

COLLABORATION POLICY

- Every student has to work individually on this assignment.
- Students are allowed to discuss high-level details about the project with others.
- Students are **not** allowed to copy the contents of a white-board after a group meeting with other students.
- Students are **not** allowed to copy the solutions from another colleague.

> ⚠ **WARNING:** All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. Plagiarism **will not** be tolerated. See CMU's **Policy on Academic Integrity** for additional information.

Last Updated: Feb 01, 2023